Automatic design of systolic chips

Patrice Quinton, Pierrick Gachet

INRIA

CENTRE DE RENNES
IRISA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél. (3) 954 90 20

Rapports de Recherche

N° 450

AUTOMATIC DESIGN OF SYSTOLIC CHIPS

Patrice QUINTON
Pierrick GACHET

Octobre 1985

AUTOMATIC DESIGN OF SYSTOLIC CHIPS

Patrice QUINTON
Pierrick GACHET
IRISA, Campus de Beaulieu
35042 RENNES - CEDEX
FRANCE
Septembre 1985

Résumé : On présente une méthode automatique pour la conception de circuits systoliques. Elle consiste à fournir des équations récurrentes uniformes correspondant à l'algorithme à réaliser. A partir de ces équations, l'ordonnancement des calculs peut être déterminé automatiquement, et des architectures systoliques peuvent être obtenues par projection. On montre aussi comment une réalisation matérielle des cellules de l'architecture peut être définie automatiquement à partir d'un jeu d'opérateurs matériels que l'on associe aux symboles fonctionnels des équations.

Abstract : An automatic method for the design of systolic chips is presented. The method is based on a description of an algorithm using Uniform Recurrent Equations. From these equations, a schedule of the computations to be carried out is derived automatically. By some simple transformations, various systolic architectures may be obtained. It is also shown how the detail of the cell implementation can be obtained automatically from a set of hardware operators associated with the functional symbols of the equations.

AUTOMATIC DESIGN OF SYSTOLIC CHIPS

Patrice QUINTON
Pierrick GACHET

IRISA, Campus de Beaulieu
35042 RENNES-CEDEX
FRANCE

1. INTRODUCTION

Recently, there has been a growing interest for systolic arrays as a particular means of implementing special-purpose VLSI architectures. Domains that could most benefit from this kind of organization are Signal Processing (S.Y. Kung, 1983 and 1984), Image Analysis or Synthesis, and Speech Processing (Frison, 1984;Charot, 1984). Most of these application domains share a nice property: the algorithms are usually expressible as numerical equations. This makes it possible to manipulate formally these algorithms using elementary transformations involving the index set of the equations. In the past, several approaches to the "extraction" of parallelism for this kind of algorithms were described by Karp et al. (1967), and Lamport (1974), in the context of SIMD-type parallelism. Recently, Moldovan (1982, 1983), Mongenet (1984) and Quinton (1983, 1984a) applied similar techniques to the particular problem of systolic array synthesis.

The DIASTOL system (Quinton, 1984b) we present here is based on the formal method described by Quinton (1983, 1984a). DIASTOL is a system whose purpose is to allow systolic architectures to be designed quickly. The design process starts from the equations of the algorithm to be implemented. These equations are entered into the system and may be transformed until they become so-called **Uniform Recurrent Equations (URE).** Then the synthesis program helps the designer building various systolic arrays by using elementary geometric transformations. This results in an abstract specification of the design, comprising the number of systolic cells, their connectivity, the timing of the data movements, and the cell functionality. The abstract specification may be used as a starting point for the functional design of each cell. Depending on the type of the calculations performed by the elementary cells, DIASTOL offers several design styles based on parallel or bit-serial operators. The hardware design process consists in associating consistently operators with the function symbols of the URE. The detailed timing of the design is then automatically found from the characteristics of these operators.

In the following paper, we describe the organization of the system. Section 2 is devoted to the basic principles of the system. The third section concentrates on the hardware cell design process. Finally, the perspective of such an approach is discussed in the conclusion.

2. PRINCIPLES OF DIASTOL

In this section, the concepts underlying the DIASTOL system are presented and examplified on the convolution product example. The interested reader may find a more formal description in Quinton (1983).

2.1. Uniform Recurrent System of Equations

The basic idea of DIASTOL is to express the algorithm to be implemented as a set of **Uniform Recurrent Equations.** The concept of URE is due to Karp et al. (1967). It provides a convenient framework for expressing a great variety of problems without any a priori knowledge of the way they can be executed. An execution scheme for the problem can be derived automatically using the properties of the URE.

Let Z^n denote the set of integer n-coordinate points of the space. An URE is a set of systems of equations, each one of which is attached to a cartesian coordinate point of a subset D of Z^n called the **domain** of the URE. To each point z of D, we attach a set of q equations having the form:

$$\forall m,\ 1 \le m \le q :$$
$$U_m(z) = f_m\left[U_{m(1)}(z-\vartheta_{m(1)}),\ \cdots,\ U_{m(p)}(z-\vartheta_{m(p)})\right]$$

where f_m is a p argument function. Vectors ϑ_i belong to a finite subset Θ of Z^n, and are called the **dependence vectors** of the URE. We impose that D be a **convex polyhedral subset** of Z^n, i.e., the points of D should verify a finite set of linear inequalities. Recall that such a set may be characterized by its vertices and rays. A point s is a **vertex** of a convex polyhedral set C if $s \in C$ and s cannot be expressed as a convex combination of other points of C. On the other hand, a vector r is a **ray** of C if, for all point x of C and all positive scalar μ, $x + \mu r$ belongs to C. Intuitively, vertices are "corners" of C, and rays are infinite directions of C. We impose, moreover that D have at most one ray.

Example 1 :

In the following, we shall illustrate all the concepts being introduced in this paper by the convolution product problem. Given a sequence $x(0)$, $x(1)$, $\cdots$, $x(i)$, ... and coefficients $w(0)$, $w(1)$, $\cdots$, $w(K)$, the convolution algorithm consists in computing the sequence $y(0)$, $y(1)$, ... , $y(i)$, ... where $y(i)$ is given by the equation:

$$y(i) = \sum_{k=0}^{K} w(k)\, x(i-k) \qquad (1)$$

This problem can be expressed as the following URE. To any integer coordinate point (i,k) lying in the domain $D = \{\ 0 \le i\ ;\ 0 \le k \le K\ \}$, we attach the following system of equations:

$$\begin{aligned}
&\forall i,\ 0 \le i\ ;\ \forall k,\ 0 \le k \le K :\\
&Y(i,k) = Y(i,k-1) + P(i,k)\\
&P(i,k) = W(i,k) \times X(i,k)\\
&W(i,k) = W(i-1,k)\\
&X(i,k) = X(i-1,k-1)
\end{aligned} \qquad (2)$$

with the following initial conditions:

$\forall i,\ 0 \le i;\ \forall k,\ 0 \le k \le K$:

$$Y(i,-1) = 0;\quad W(-1,k) = w(k);\quad X(i-1,-1) = x(i);\quad X(-1,k-1) = 0$$

It can be easily checked that $y(i) = Y(i,K)$. This URE can be represented as a graph such as that of Fig. 1. where the points represent computations and arrows represent the dependences between the computations.
End of example.

It should be pointed out that in most cases, obtaining a URE from the equations of a problem is easily achieved by means of elementary algebraic transformations.

2.2. Automatic synthesis of systolic arrays

The system of equations that is associated with a particular point z represents a calculation to be done in order to solve the problem. We suppose first that the system of equations attached to a given point z is an **atomic** computation, in the sense that all the arguments $U_{m(j)}$ that appear in the right-hand side of a equation should have been computed before all the results $U_m(z)$ are evaluated, and also that the computation of any equation takes exactly one unit of time. This hypothesis will be removed later on when the detail of the cell implementation will be examined.

The task of designing any parallel device implementing the URE consists in solving two problems:

- order the calculations consistently with the dependences, by means of a **timing function** t from Z^n to Z. The value $t(z)$ is the time at which computation at point z is performed.

- map the calculations on a finite set of processors without conflict. This is done by defining an **allocation function** α from Z^n to a finite set. The value $\alpha(z)$ identifies the processor on which z is to be computed.

It is thus natural to impose the following constraints on the timing and allocation functions:

- the time at which an equation is calculated must be greater than the time at which its arguments are available, augmented by the time needed to perform the computation. This gives:

$$\forall z,\ \forall \vartheta \in \Theta : t(z) \ge t(z - \vartheta) + 1 \qquad (3)$$

- secondly, the computation should start once (recall that D may be infinite, as it is the case in the convolution product example). This is of course true if t is restricted to be positive:

$$\forall z \in D: t(z) \ge 0 \qquad (4)$$

- the number of points to be computed at a given time instant τ should be bounded. If we denote as N the set of non-negative integers, we should have:

$$\exists M \in N,\ \forall \tau \ge 0:\ ||\{z \mid t(z) = \tau\}|| \le M \qquad (5)$$

where $||\ E\ ||$ denotes the cardinality of the set E.

- two different points x and y of D that are mapped on the same processor should not be computed at the same instant, i.e.:

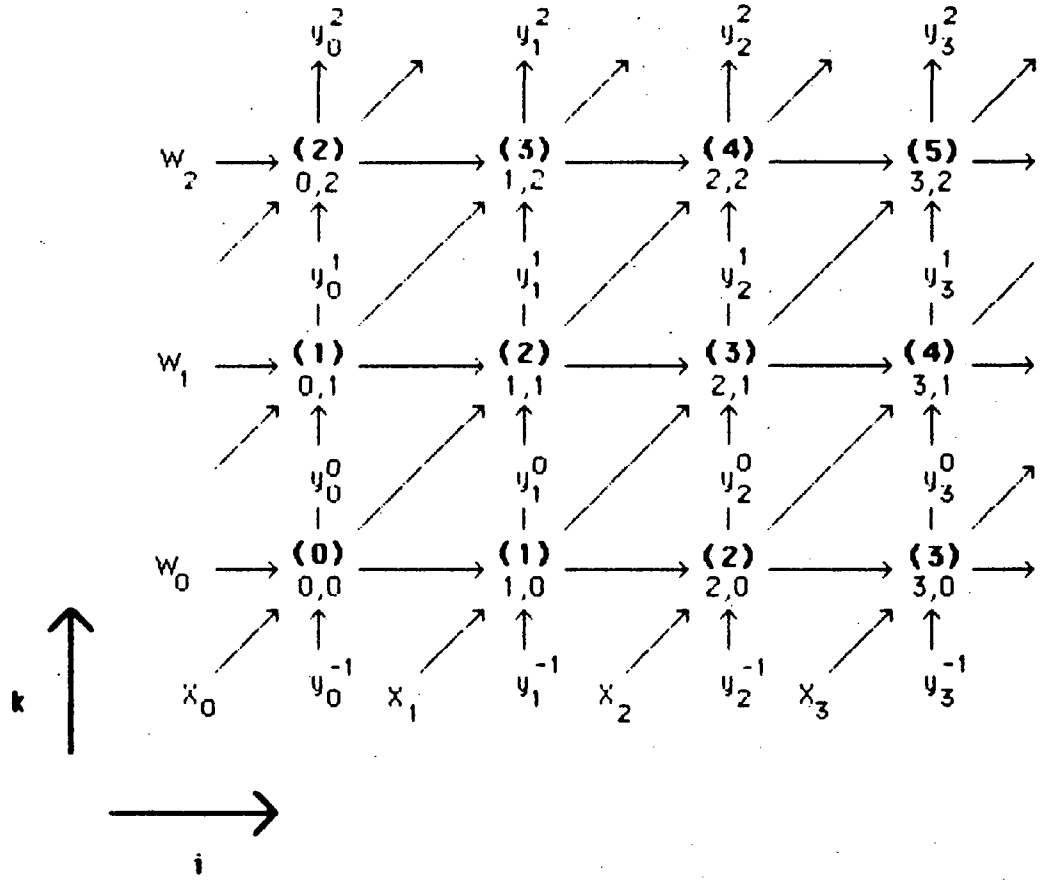


Fig. 1: Dependence graph for the convolution product. The domain D of the corresponding URE is an infinite band defined by $i \geq 0$ and $0 \leq k \leq K$. D has two vertices $(0,0)$ and (O,K), and a ray $(O,1)$. The timing function $t(i,k) = i+k$ is indicated in boldface numbers.

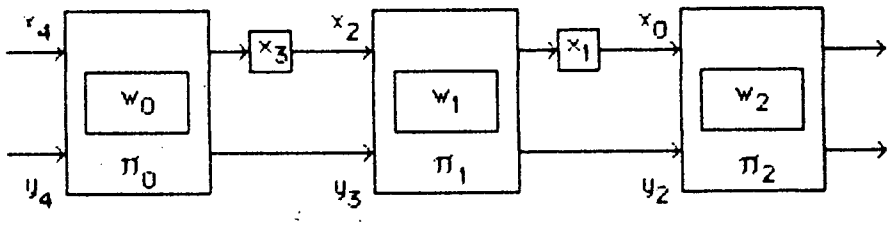


Fig. 2: Systolic array for the convolution product, as obtained by the allocation function $a(i,k) = k$

$$\forall x, y \in D: \quad a(x) = a(y) \rightarrow t(x) \neq t(y) \tag{6}$$

- finally, the number of processors must be finite, or:

$$|| \{ a(z) \mid z \in D \} || \text{ is finite} \tag{7}$$

In order to obtain systolic architectures, we constraint t to be an **affine** function from Z^n to Z, i.e. t has the form $t(z) = \lambda^T z + \alpha$ where $\lambda = (\lambda_1, \cdots, \lambda_n)^T$ is a vector of Z^n and α is an integer ($\lambda^T z$ stands for the dot product of λ and z). Moreover, we restrict ourselve to the case when a is a projection from Z^n to Z^{n-1} along a given vector $u \in Z^n$, called the **projection vector.** The function a is thus a linear mapping a from Z^n to Z^{n-1} such that:

$$\forall x \in Z^n, \forall \mu \in R: a(x + \mu u) = a(x)$$

The following theorem gives conditions upon which λ, α and u define a systolic design:

Theorem 1:

Let S and R be respectively the set of vertices and rays of the domain D. Then λ, α and u define a systolic design if:

(i) $\forall \vartheta \in \Theta, \lambda^T \vartheta \geq 1$

(ii) if D has a ray r, then $\lambda^T r > 0$ and vectors u and r are colinear

(iii) $\forall s \in S: \lambda^T s + \alpha \geq 0$

(iv) $\lambda^T u > 0$,

Sketch of the proof:

A detailed proof may be found in Quinton (1983). Condition (i) corresponds obviously to (3) and the fact that t is affine. Condition (ii) expresses that t must have a lower bound on D, that $a(D)$ is bounded and that there is a finite number of points having the same timing value. Condition (iii) is related to the fact that t must be positive. Finally, (iv) implies that two points x and y that are mapped on the same processor have different timing values $t(x)$ and $t(y)$. **QED.**

Theorem 1 gives a constructive means of finding systolic arrays for the following reasons. It defines a set of linear inequalities involving λ and the scalar α. It is thus possible to find out automatically whether there exists a solution or not. Moreover, each of the parameters λ_j must be taken in such a way that the inequalities should be met as tightly as possible. If we denote as Λ the subset of Z^n which is defined by (i), (ii), and (iii), this means that vertices of Λ are the only candidates to be considered, since they define solutions for which the maximum number inequalities become actually equalities. Very often in practice, Λ has only one vertex which represents the fastest solution. When there are several vertices, the choice depends on what parameter (delay or throughput) of the design should be optimized (see Quinton, 1983). Once a timing function is found, any choice of a projection vector u that meets the conditions of the theorem gives a systolic array. The detail of the calculation of a is not given here, but is rather simple as it consists in a change of basis over Z^n.

Given t and a, the structure of the architecture is defined as follows. The cells of the array are the points of the projected convex domain $a(D)$. Each cell $\pi \in a(D)$ is considered as one single functional element. Its operation results

from the equations of the URE, the timing-function and the allocation-function. First, note that, since a is linear, we have:

$$\forall z, \forall \vartheta \in \Theta: a(z - \vartheta) = a(z) - a(\vartheta).$$

Thus, cell π takes its input $U_{m(j)}$ from cell $\pi - a(\vartheta_{m(j)})$ (or receives $U_{m(j)}$ from an external memory whenever $\pi - a(\vartheta_{m(j)})$ does not belong to $a(D)$). The time between the calculation of $U_{m(j)}(z - \vartheta_{m(j)})$ and its use is:

$$t(z) - 1 - t(z - \vartheta_{m(j)}) = \lambda^T \vartheta_{m(j)} - 1$$

This quantity defines the number of delay elements that should be placed between the two cells, for the communication of this argument.

Example 1 (continued) :

Consider again the convolution product. From (i), we obtain the following inequalities:

$$\lambda_2 \geq 1 \; ; \; \lambda_1 \geq 1 \; ; \; \lambda_1 + \lambda_2 \geq 1$$

On the other hand, the domain D has two vertices, (0,0) and (0,K), and one ray, (1,0). From (ii) and (iii), we obtain:

$$\lambda_1 > 0 \; ; \; \alpha > 0 \; ; \; K\lambda_2 + \alpha \geq 0$$

The only vertex of the convex set defined by this system is:

$$\lambda_1 = 1 \; ; \; \lambda_2 = 1 \; ; \; \alpha = 0$$

The resulting timing function, as depicted by Fig. 1, is $t(i,k) = i+k$.

The only way to choose u is to take it parallel to the i-axis, since D has one ray. The projection vector is $u = (1,0)$. This leads to the allocation function $a(i,k) = k$. The resulting domain $a(D)$ is the segment $[(0,0),(0,K)]$. In other words, the systolic array that is obtained has $K+1$ processors.

Fig. 2 depicts the systolic array that results from the above defined URE, timing and allocation functions. This design was described by Kung (1982). This array has $K+1$ cells numbered π_0 through π_k. Coefficients w_k stay in cell π_k as $a(\vartheta_w) = 0$. Values $Y(i,k)$ and $X(i,k)$ flow from cell π_k to π_{k+1} as $a(\vartheta_Y) = a(\vartheta_X) = 1$. However, the timing of the Y's and the X's are different, since $\lambda^T \vartheta_Y = 1$ whenever $\lambda^T \vartheta_X = 2$. This explains why the X's have to be delayed between two consecutive cells.

End of example.

3. DESIGNING A CELL

The previous section provides a convenient method to derive quickly systolic arrays. This section describes the detailed design of each cell, taking into account the hardware implementation aspects of a chip design. The approach that is taken in DIASTOL makes use of a set of predefined operators, which are characterized by a few parameters. Such operators are used to replace corresponding function symbols of the equations. The timing and allocation functions that have been defined in section 2 are extended in order to allow the delays introduced by the operators to be computed automatically. Depending on the complexity of the functions of the URE, several hardware design styles could be used. Here, we shall consider two different types of operators: parallel

operators and bit-serial operators. In the first subsection, we describe a simple hardware operator model. Examples of operators for the multiplication of binary numbers are given in subsection two. The third subsection gives a theoretical result that allows a cell to be designed from a set of operators.

3.1. Hardware operator modelization

For the purpose of this paper, we modelize an hardware operator as a "black box" Op (see Fig. 3) having several input ports I_i and one output port O (multi-output operators are not considered here for the sake of simplicity). Associated with the operator is a function symbol $f(Op)$ and several attributes that describe the timing behaviour of the operator. We assume that this timing behaviour is expressed relatively to a reference time, in term of discrete time steps such that clock cycles or phases. Each operator is characterized by its type, **parallel** or **bit-serial**, and three timing parameters, called its **latency** $\delta(Op)$, its **periodicity** $\psi(Op)$, and its **bit-skew** $\sigma(Op)$.

These parameters have the following meaning. The input values enter simultaneously the operator starting from the reference time. The input bits enter, and the output bits leave the operator in parallel or bit-serially, least significant bit (LSB) first. The latency of the operator is then the number of time steps between the input of the LSB of the input values and the output of the LSB of the result. The periodicity of the operator is the number of time steps between two consecutive use of the operator. Finally, the bit-skew is the delay that separates the input or output of two successive bits of a value.

In a parallel operator, each input port I_i is $n(I_i)$ bit wide, and the output port is $n(O)$ bit wide. In a bit-serial operator, each port is only one bit wide, which receives (or delivers) in turn the input or output value bits. The periodicity of a bit-serial operator must therefore be greater than the time necessary to enter and deliver the values.

More precisely, let $x_1, \cdots, x_n$ the input values of an operator. Then:

- input I_i receives the k^{th} bit of an input value x_i at time step $k \times \sigma(Op)$ after the reference time,

- output O provides the k^{th} bit of $f(Op)[x_1, \cdots, x_n]$ at time $\delta(Op) + k \times \sigma(Op)$,

- the operator may be reused for another computation $\psi(Op)$ time steps after the reference time.

A more general model could be derived in order to cover the case when the inputs and the outputs are not used at the same rate, but this would complicate unnecessarily the following explanations.

With the above definition, a **purely parallel operator** is modeled as an operator where $\sigma(Op) = O$. The operator has $\frac{\delta(Op)}{\psi(Op)}$ pipeline stages. The case of a non pipelined operator is covered by the particular case when $\delta(Op) = \psi(Op)$.

We call **parallel bit-slice operator** a parallel operator such that $\sigma(Op) \neq 0$. The name bit-slice refers here to the fact that the computation is done by slices of hardware one bit after the other, but in parallel.

3.2. Examples of operators

In this subsection, we show how binary multipliers can be derived from their equation using the method described in section 2, and how they can be abstracted using the hardware model. These examples will be used later for designing a convolution systolic array. Consider the multiplication of two positive numbers U and T. $U = U_{q-1}U_{q-2}\cdots U_0$ has q bits and $T = T_{p-1}\cdots T_0$ has

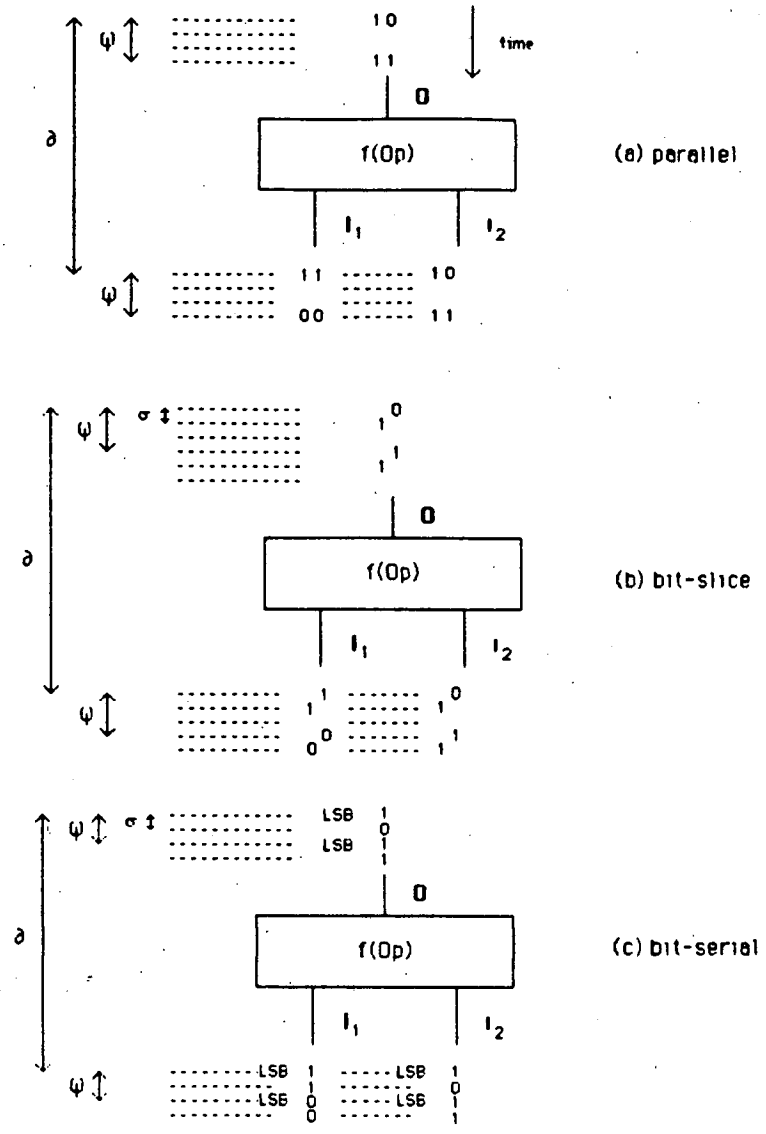


Fig. 3: Simple operator model.
(a) Parallel operator with latency δ and periodicity $\psi = 3$.
(b) Bit-slice parallel operator with skew $\sigma = 1$.
(c) Bit-serial operator.

p bits. The result is $V = V_{r-1} \cdots V_0$ and has $r = p+q$ bits. In order to build a parallel and a bit-serial multiplier, let us consider the diagram of Fig. 4. With each point of the domain $D = \{ 0 \le j \le p \ ; \ i \ge 0 \ ; \ 0 \le i+j \le r-1 \}$, we associate the following calculation:

$$P(i,j) = FA\_SUM\Big(P(i-1,j+1),\ AND\big(U(i,j-1), T(i-1,j)\big),\ C(i-1,j) \Big)$$
$$C(i,j) = FA\_CARRY\Big(P(i-1,j+1),\ AND\big(U(i,j-1), T(i-1,j)\big),\ C(i-1,j) \Big)$$

where *FA_SUM* and *FA_CARRY* denote respectively the functions producing the sum and the carry of a full-adder cell. Assuming that U is extended to the MSB by zeros up to r bits, and that the following initial conditions hold:

$T(-1,j) = T_j$ for all j and $U(i,-1) = U_i$ for all i

$P(-1,j) = 0$ and $P(i,p+1) = 0$ for all i,j

$C(-1,j) = 0$ for all j

then it can be easily checked that the product bits are given by:

$P_i = P(i,0)$ for all i

In such a scheme, the bits of the product are summed along the lines $i+j=r$ and the carries are propagated along lines parallel to axis i. Assuming a conventional two phase non overlapping clock scheme, the duration of the function evaluation is one clock phase. Applying theorem 1 gives the timing function $t(i,j) = 2i+j$ illustrated by Fig. 4.

A bit-serial multiplier is obtained by projecting the domain D along axis i. Fig. 5a shows the result. The multiplier has q cells. Bits U_i flow from bottom to top and enter a new full-adder cell every clock-phase. Each bit T_j of the multiplier stays in cell j of the operator. Bits P_i of the product are produced by the cell from top to bottom. Notice that the full-adder cell should be provided with a reset circuitry (not represented here) in order to set the carry and product bits to 0 when U_0 enters the cells.

This operator may be abstracted according to our model as a bit-serial operator with latency 1, bit-skew 2 and periodicity $2r$. The timing parameter are easily obtained from the diagram of Fig. 4. This design could be extended in order to handle two's complement binary numbers using for example Booth's algorithm. Note that our design is similar to the well-known bit-serial algorithm of Lyon (Lyon, 1976; Newkirk, 1983), except that the latency is independent from the length of the multiplicand. Lyon's multiplier would be obtained by changing the orientation of the P flow in the diagram of Fig. 4.

The diagram of Fig. 4 may also be used directly in order to obtain a parallel bit-slice operator. The result is depicted by Fig. 5b. This operator has the same latency and bit-skew as the bit-serial version, but its periodicity is 2, as one data may enter the multiplier every other clock phase.

3.3. Design of systolic cells

The basic principle of our methodology is to associate a hardware operator with each functional symbol of the URE. We consider first the case when these operators have no bit-skew, then we extend the method to skewed operators.

3.3.1. Non skewed operators

Consider a URE having q equations:

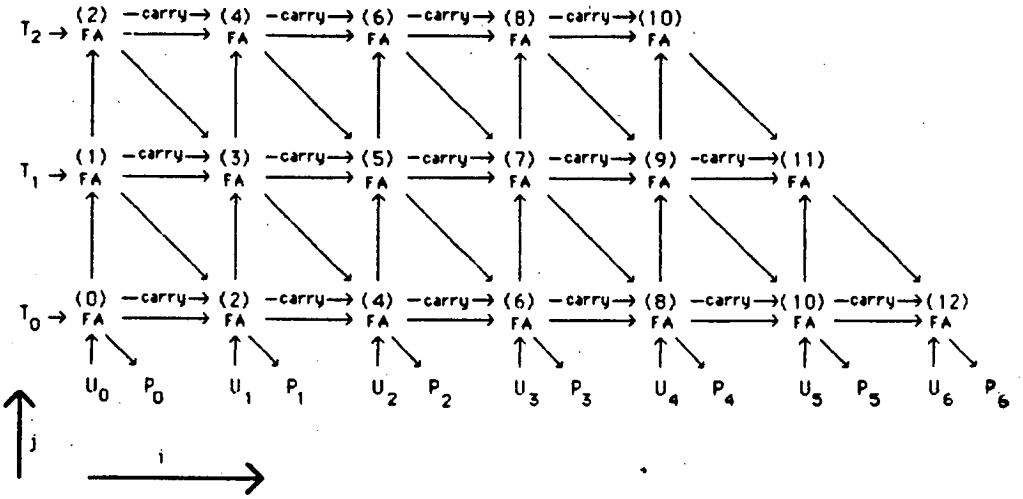


Fig. 4: Dependence graph for a multiplier $P = UT$. U and T are 3 bit wide.

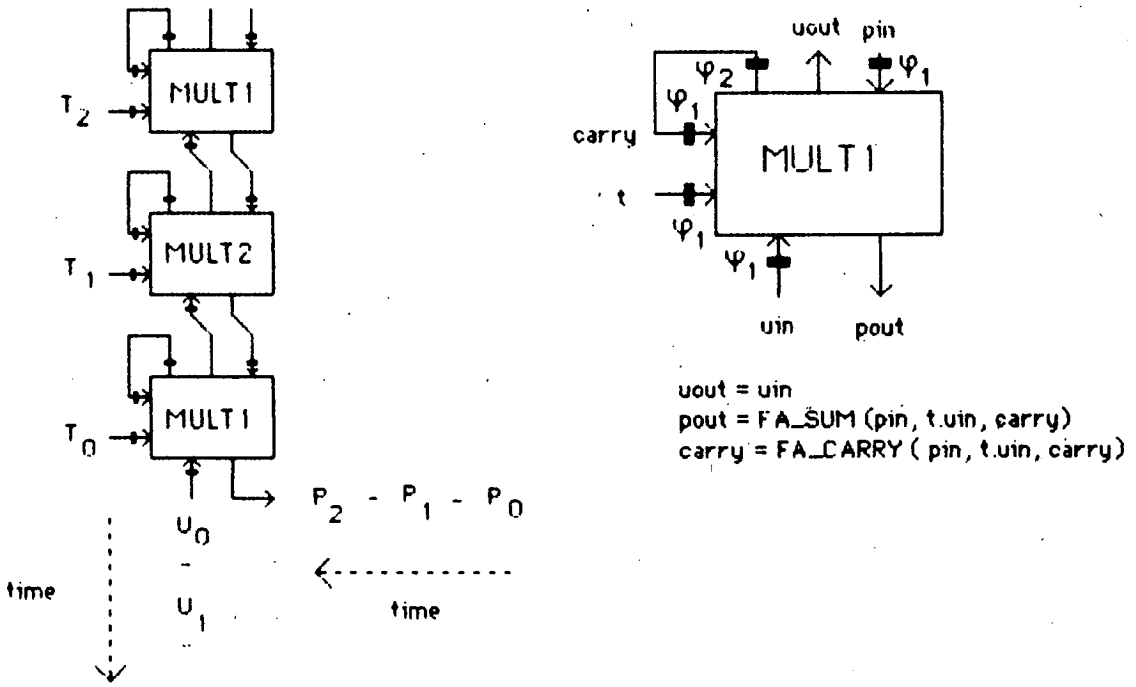
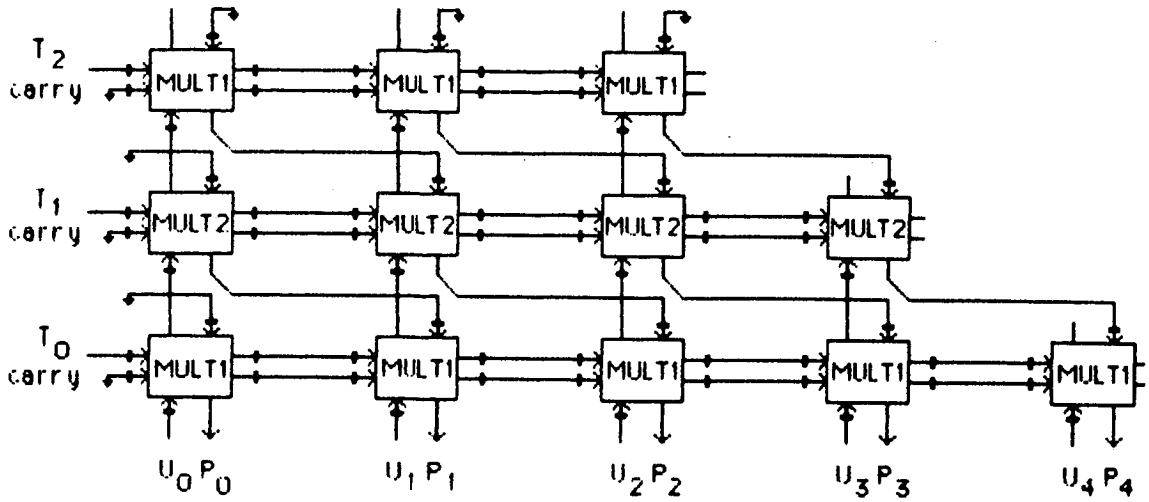


Fig. 5a: A bit-serial multiplier, with latency 1, periodicity $2r$, and bit-skew 2 (MULT2 is the same cell as MULT1, but clocked on the opposite clock phases).



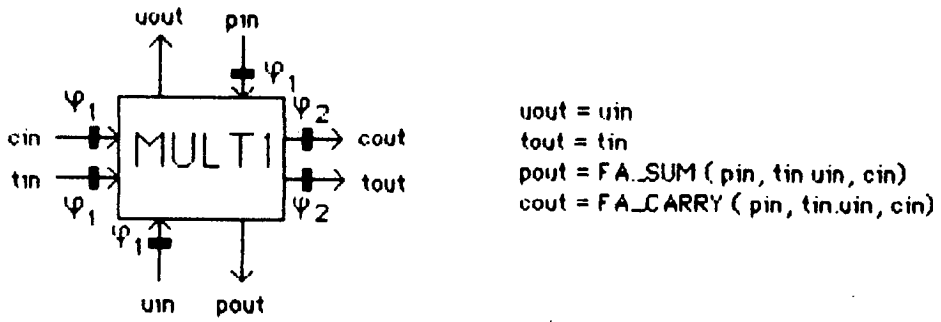


Fig. 5b: A parallel multiplier with latency 1, periodicity 2 and bit-skew 2 (MULT2 is as MULT1, but clocked on the opposite clock phases).

$$U_m(z) = f_m\left[U_{m(1)}(z-\vartheta_{m(1)}) \, , \cdots \, , U_{m(p)}(z-\vartheta_{m(p)})\right]$$

According to the above principle, each cell has q hardware operators, each one corresponding to one equation. The timing of the computation in each cell results from the following rule, which replaces the brute-force approach taken in section 2. Each individual equation is considered to be an **atomic** expression: its left-hand side term is computed only when all the arguments needed for the right-hand side calculation are available. The equations attached to a given point z are not computed at the same time as it was the case in section 2. However, the relative schedule of these equations are kept constant over D. In other words, the time at which equation m_1 of the URE is computed differs from the time of equation m_2 by a constant.

In order to modelize this new rule, we now attach one particular timing-function $t_m(z) = \lambda^T z + \alpha_m$ to each equation that defines when $U_m(z)$ is obtained. Note that all these functions have the same parameter λ. The **schedule** of the computation is now defined by a $n+q$ dimensional vector $(\lambda \, , \alpha_1 \, , \ldots \, , \alpha_q)^T$.

In order to design the cell, let us associate a parallel operator Op_m having latency δ_m and periodicity ψ_m, with each function f_m. We assume that the operators are assigned consistently, i.e. that $f\,[Op_m] = f_m$ for all m, and that an argument U_m is carried out by input or output ports having the same number of bits, and the same periodicity.

Let $(\lambda, \alpha_m \, , \cdots \, , \alpha_q)^T$ be a schedule and u be a projection direction. The constraints that must be satisfied may be summarized as follows:

- since $U_m(z)$ is computed when all the arguments needed are available, and since the operator associated with f_m has a latency δ_m, $t_m(z) \geq t_{m(j)}(z - \vartheta_{m(j)}) + \delta_m$ for all j,

- on the other hand, the timing-function and the allocation function should be such that the operators are used at a rate that is consistent with their periodicity. In other words, two calculations that are mapped on the same cell must be separated by enough time steps so that the operators are not over-used.

The following theorem gives a sufficient condition for the existence of a schedule and of an allocation function:

Theorem 2:

Let S and R be respectively the set of vertices and rays of the domain D. Assume that parallel operators Op_m are associated consistently with the functions f_m of the URE. Then the schedule $(\lambda,\alpha_1 \, , \cdots \, , \alpha_q)^T$ and the projection vector u define a systolic array if:

(i) $\forall m, \forall j \, : \, \lambda^T \vartheta_{m(j)} + \alpha_m - \alpha_{m(j)} \geq \delta_m$

(ii) if D has a ray r, then $\lambda^T r > 0$, and u and r are colinear vectors.

(iii) $\forall s \in S$: $\lambda^T s + \alpha_m \geq 0$,

(iv) $\lambda^T \dfrac{u}{GCD(u_1, \cdots, u_n)} \geq \max_{m=1}^{q} \psi_m$

Sketch of the proof:

Condition (i) express the new rule concerning the evaluation of the URE. Conditions (ii) and (iii) are the same as in theorem 1. Condition (iv) neads more explanations. The vector $\bar{u} = \dfrac{u}{GCD(u_1, \cdots, u_n)}$ is the **generator** vector for the

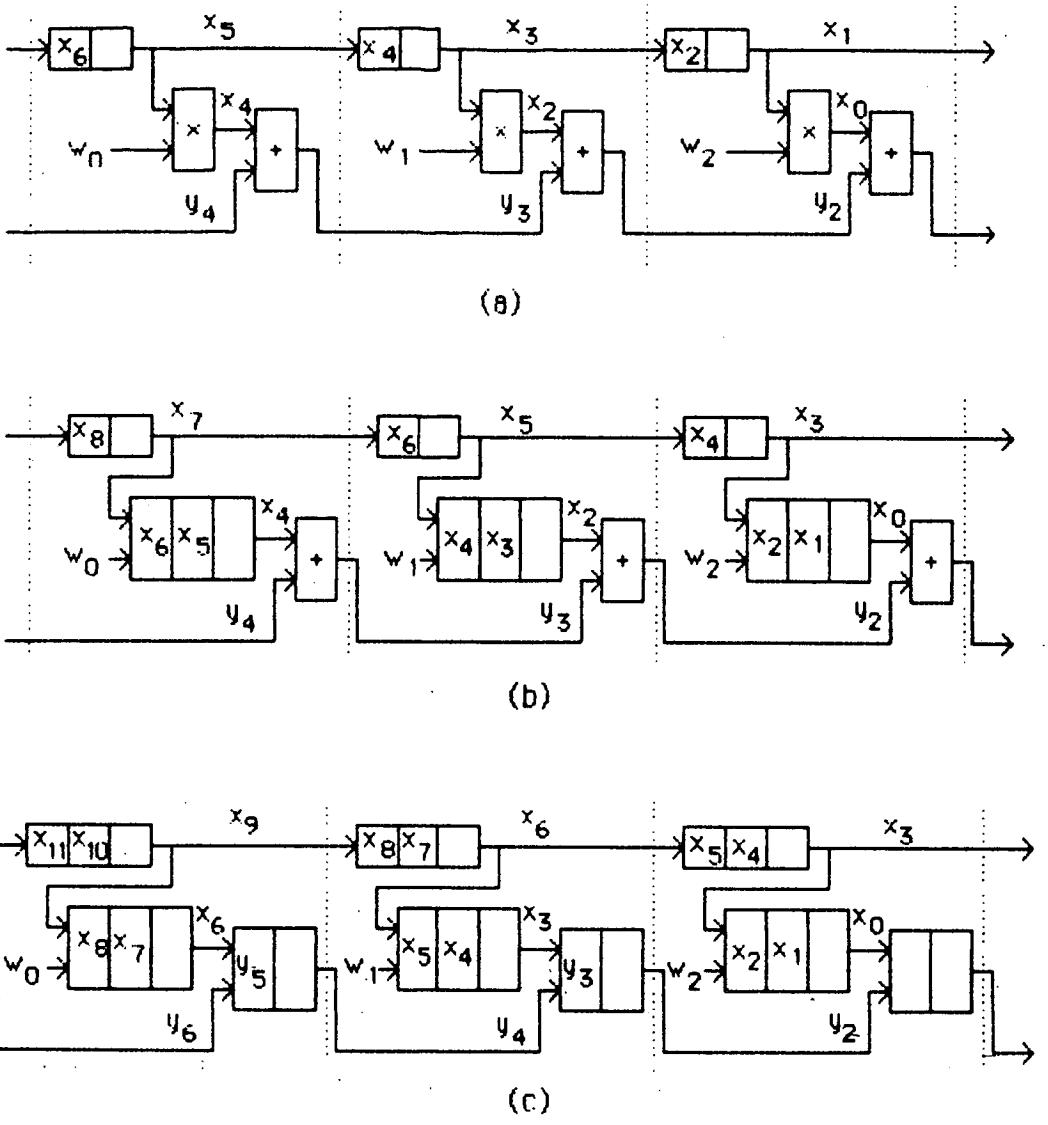


Fig. 6: Two-level pipelined systolic arrays for the convolution product.
(a) The add and multiply operators each take one time step.
(b) The multiply operator is pipelined (latency 3, periodicity 1).
(c) The add operator is pipelined (latency 2).

integer coordinate points of any line parallel to u. In other words, if L is a line of R^n parallel to u, and if x_0 is an integer coordinate point of L, then $L \cap Z^n = \{ x_0 + k\bar{u} \mid k \in Z \}$. As a consequence, the quantity $\lambda^T \frac{u}{GCD(u_1, \cdots, u_n)}$ is the periodicity of the systolic array that should be greater than the periodicity of the operators. **QED.**

These four conditions may be used practically to synthesize a systolic array, provided the functions f_m may be directly mapped on hardware operators.

Example 2:

Consider again the convolution product, and suppose we want to implement the add and multiply functions using off-the-shelf parallel pipelined components. Assume first that both components have latency 1 and periodicity 1. Note that the identity function is supposed to have delay 1 and periodicity 1, in order to avoid the broadcasting of data. This hypothesis is however not compulsory, and if removed, would possibly result in what is called a semi-systolic design by Kung (1982), where some data are broadcasted to all the cells. Denote as α_Y, α_P, α_W, and α_X, the scalars associated with the timing function of Y, P, W, and X respectively. Applying theorem 2, we have, from (i):

$$\lambda_2 \geq 1 \; ; \; \alpha_Y - \alpha_P \geq 1 \; ; \; \alpha_P - \alpha_X \geq 1 \; ; \; \alpha_P - \alpha_W \geq 1 \; ; \; \lambda_1 + \lambda_2 \geq 1 ; \; \lambda_1 \geq 1$$

From (ii):

$$\lambda_1 > 0$$

From (iii):

$$\alpha_Y \geq 0 \; ; \; \alpha_P \geq 0 \; ; \; \alpha_W \geq 0 \; ; \; \alpha_X \geq 0 \; ;$$

$$K\lambda_2 + \alpha_Y \geq 0 \; ; \; K\lambda_2 + \alpha_P \geq 0 \; ; \; K\lambda_2 + \alpha_W \geq 0 \; ; \; K\lambda_2 + \alpha_X \geq 0 \; ;$$

Finally, from (iv):

$$\lambda_1 \geq 1$$

The only vertex of the convex domain defined by these equations is given by:

$$\lambda_1 = 1 \; ; \; \lambda_2 = 1 \; ; \; \alpha_X = \alpha_W = 0 \; ; \; \alpha_P = 1 \; ; \; \alpha_Y = 2$$

The resulting array is depicted by Fig. 6a. This solution is to be compared with that of Fig. 2. The only difference lies in the fact that the y flow is delayed by one units of time in order to be synchronized with the availability of the product $w.x$.

Now assuming a three stage pipelined multiplier, i.e., with latency 3 and periodicity 1, constraint (i) becomes:

$$\lambda_2 \geq 1 \; ; \; \alpha_Y - \alpha_P \geq 1 \; ; \; \alpha_P - \alpha_X \geq 3 \; ; \; \alpha_P - \alpha_W \geq 3 \; ; \; \lambda_1 + \lambda_2 \geq 1 ; \; \lambda_1 \geq 1$$

and the solution is (see Fig. 6b):

$$\lambda_1 = \lambda_2 = 1 \; ; \; \alpha_X = \alpha_W = 0 \; ; \; \alpha_P = 3 \; ; \; \alpha_Y = 4$$

In other words, the periodicity of the design has not been changed, and maybe

more surprinsigly, the delays on the x lines are the same. Moreover, as in the version of Fig. 6a, the y flow has been delayed.

Finally, consider the case when the adder has two pipeline stages. Applying theorem 2 gives the following new constraints:

$$\lambda_2 \geq 2 \; ; \; \alpha_Y - \alpha_P \geq 2 \; ; \; \alpha_P - \alpha_X \geq 3 \; ; \; \alpha_P - \alpha_y \geq 3 \; ; \; \lambda_1 + \lambda_2 \geq 1; \; \lambda_1 \geq 1$$

and the new solution is (Fig. 6c):

$$\lambda_1 = 1 \; ; \; \lambda_2 = 2 \; ; \; \alpha_X = \alpha_y = 0 \; ; \; \alpha_P = 3 \; ; \; \alpha_Y = 5$$

The parameter λ_2 is now modified, as the delay between the input of a y value and its output becomes 6 instead of 3 in the previous versions. The periodicity of the desgn is thus doubled. **End of example.**

This example shows that Theorem 2 gives a very powerful means to compute the exact timing of the data inside the cell, which is not at all obvious when there is a large number of operators.

3.3.2. Skewed operators

When designing a systolic chip, it may be interesting to implement the basic arithmetic operations themselves using "systolic like" techniques. The usual implementation of an arithmetic unit for instance has the main drawback that the basic cycle duration must allow for the propagation of the carry. Even when using carry-lookahead or precharging techniques (Mead, 1980), this tends to slow down significantly the clock period of the chip. An alternative to these conventional implementation techniques consists in using bit-slice or bit-serial operators. In this section, we show how theorem 2 may be extended in a straightforward manner in order to obtain the timing of systolic implementations based on such operators.

Suppose that we associate consistently with each functional symbol an operator possibly with a non-zero bit-skew. The following corollary gives the timing of the corresponding design:

Corollary 1 :

Under the above defined assumptions, if $(\lambda, \alpha_1, \cdots, \alpha_q)^T$ is a schedule for a URE in the sense of Theorem 2 and u is a projection vector, then the time $t_m(z,p)$ at which bit p of argument $U_m(z)$ is produced is given by:

$$t_m(z,p) = \lambda^T z + \alpha_m + \sigma_m p$$

The proof is obvious and left to the reader. This corollary gives a straightforward means of implementing a systolic cell simply by replacing each functional symbol by any type of operator.

Example 3:

We consider here the systolic array that results of the use of parallel bit-slice adders and multipliers. Fig. 7 shows a straightforward bi-slice parallel adder with latency 1, periodicity 2 and bit-skew 2. Together with the parallel bit-slice multiplier depicted by Fig. 5b, these operators could be assigned consistently with the URE of the convolution product. Applying Theorem 2 gives the following schedule for Y, P, and X:

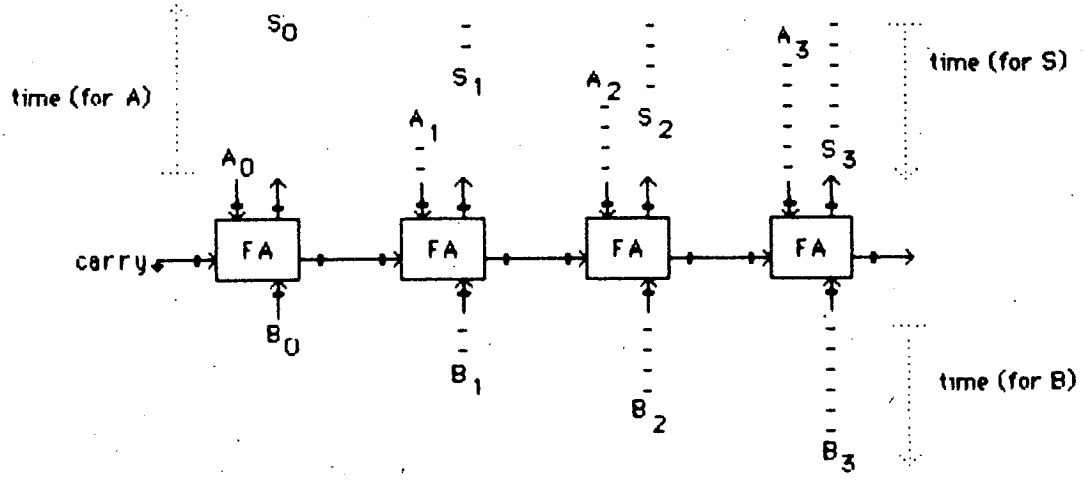


Fig. 7: Parallel bit-slice adder $S = A+B$ (latency 1, periodicity 2 and bit-skew 2).

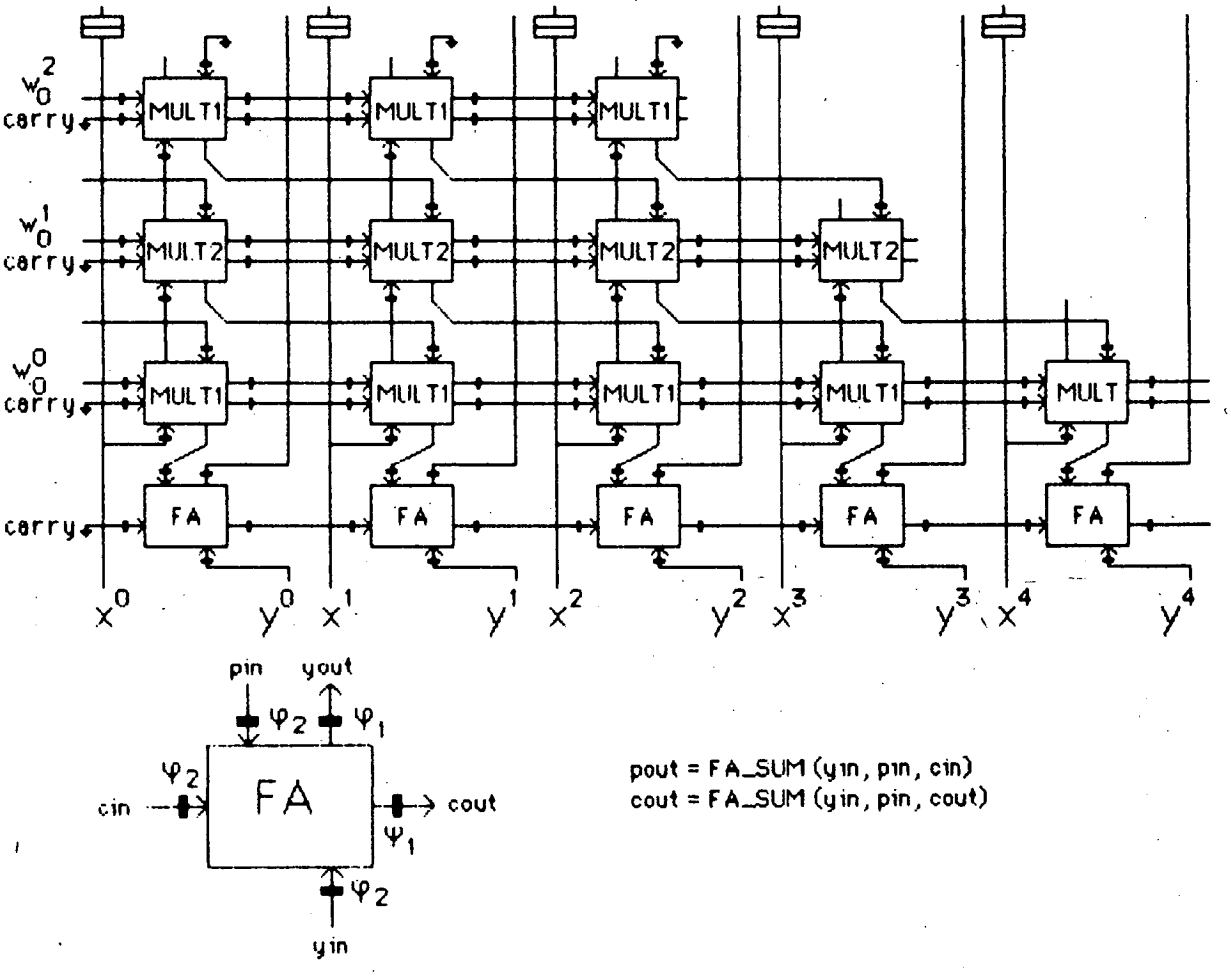


Fig. 8: A cell of a systolic convolver, as implemented using parallel bit-slice operators.

$$t_Y(i,k,p) = 2i + k + 2p + 2$$
$$t_P(i,k,p) = 2i + k + 2p + 1$$
$$t_X(i,k,p) = 2i + k + 2p$$

The corresponding implementation is depicted by Fig. 8.

A bit-serial implementation may be similarly derived by using the serial adder of Fig. 9 together with the serial multiplier of Fig. 5a. Applying Corollary 1 to the URE defining the convolution product, we obtain:

- from (i) of Theorem 2:

$$\lambda_2 \geq 1 \; ; \; \alpha_Y - \alpha_P \geq 1 \; ; \; \alpha_P - \alpha_X \geq 1 \; ; \; \alpha_P - \alpha_W \geq 1 \; ; \; \lambda_1 + \lambda_2 \geq 1 ; \; \lambda_1 \geq 1$$

- from (ii):

$$\lambda_1 > 0$$

- from (iii):

$$\alpha_Y \geq 0 \; ; \; \alpha_P \geq 0 \; ; \; \alpha_W \geq 0 \; ; \; \alpha_X \geq 0 \; ;$$

$$K\lambda_2+\alpha_Y \geq 0 \; ; \; K\lambda_2+\alpha_P \geq 0 \; ; \; K\lambda_2+\alpha_W \geq 0 \; ; \; K\lambda_2+\alpha_X \geq 0 \; ;$$

- finally, from (iv):

$$\lambda_1 \geq 2r$$

The unique vertex solution to this problem is:

$$\lambda_1 = 2r \; ; \; \lambda_2 = 1 \; ; \; \alpha_X = \alpha_W = 0 \; ; \; \alpha_P = 1 \; ; \; \alpha_Y = 2$$

The resulting array is depicted by Fig. 10. The main structural difference between this design and that of Fig. 2 is the $2r+1$ time unit delay on the X line, which indeed comes from the fact that the operators are bit-serial.

4. CONCLUSION

In this paper, we have shown how systolic architectures may be designed automatically from their specification using Uniform Recurrent Equations. Moreover, it has been shown how the detailed design of the cells of the architecture can be obtained by associating hardware operators with the functional symbols of the URE. A simple hardware model that helps abstracting the operators has been presented. The DIASTOL system is based on the above described method. It offers interactive graphics facilities for the visualization of the domain of the URE and the systolic arrays that are designed. Further work must be done in order to characterize what class of algorithms can be represented using URE. Formal transformations should also be provided, in such a way that the method can cope as far as possible with the usual equations of the algorithms. A more complete hardware operator model could also be defined, in order to cover the building blocks found in a VLSI library.

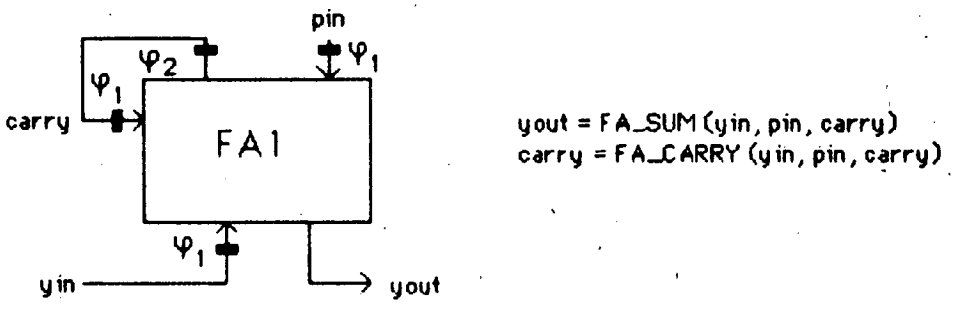


Fig. 9: A bit-serial adder.

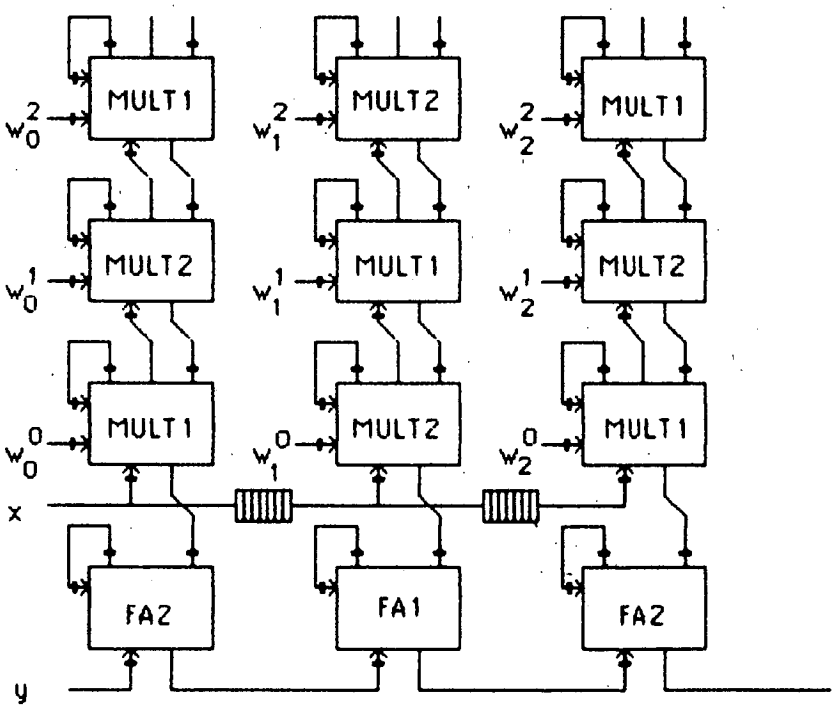


Fig. 10: Bit-serial implemention of the convolution product.

5. REFERENCES

(Charot, 1984)
F. Charot, P. Frison, P. Quinton, "Systolic Architectures for Connected Speech Recognition", IRISA Research Report, No 234, August 1984.

(Frison, 1984)
P. Frison, P. Quinton, "A VLSI Parallel Machine for Speech Recognition," in *Proc. of ICASSP 84*, pp. 25B.3.1-25B.3.4, 1984.

(Karp, 1967)
R.M. Karp, R.E. Miller, S. Winograd, "The Organization of Computations for Uniform Recurrence Equations," *JACM*, Vol. 14, NO. 3, juillet 1967, pp. 563-590.

(Kung, 1981)
H.T. Kung, L.M. Ruance, D.W.L. Yen, "A Two-Level Pipelined Systolic Array for Convolutions," in *VLSI Systems and Computations*, ed H.T. Kung, R.R. Sproull, G. Steele, Jr., Computer Science Press, pp. 255-264, 1981.

(S.Y. Kung, 1983)
S.Y. Kung, Y.H. Hu, "A Highly Concurrent Algorithm and Pipelining Architecture for Solving Toeplitz Systems," *IEEE Trans. on ASSP*, Vol. ASSP-31, No. 1, pp. 66-76, Feb. 1983.

(S.Y. Kung, 1984)
S. Y. Kung, "On Supercomputing with Systolic/Wavefront Array Processors," *Proceedings of the IEEE*, Vol 72, No. 7, July 1984, pp. 867-884.

(Kung, 1982)
H.T. Kung, "Why Systolic Architectures? ," *Computer*, Vol. 15, No 1, Jan 1982, pp. 37-46.

(Lamport, 1974)
L. Lamport, "The Parallel Execution of DO Loops," *CACM*, Vol 17, Nr 2, February 1974, pp 83-93.

(Lyon, 1976)
R.F. Lyon, "Two's Complement Pipeline Multipliers," *IEEE Trans. Comm.*, Vol. COM-24, pp. 418-425, April 1976.

(Mead, 1980)
C. Mead, L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980.

(Moldovan 1982)
D. Moldovan "On the Analysis and Synthesis of VLSI Algorithms," *IEEE Trans. on computers*, Vol. C-31, No 11, November 1982

(Moldovan 1983)
"On the Design of Algorithms for VLSI Systolic Arrays, D. Moldovan "On the Design of Algorithms for VLSI Systolic Arrays," *Proceedings of the IEEE*, Vol. 71, No 1, January 1983

(Mongenet, 1985)
C. Mongenet, "SYSTOL: un logiciel de conception d'algorithmes systoliques," CRIN Research Report, No 85-R-003, Jan. 1985.

(Newkirk, 1983)
J. Newkirk, R. Mathews, *The VLSI Designer's Library*, Addison-Wesley, 1983.

(Quinton, 1983)
P. Quinton, "The Systematic Design of Systolic Arrays," IRISA Research Report, No 193, march 1983.

(Quinton, 1984a)
P. Quinton, "Automatic Synthesis of Systolic Arrays from Recurrent Uniform Equations," *11th Annual Int. Symp. Computer Arch.*, June 1984, Ann Arbor, pp. 208-214.

(Quinton, 1984b)
P. Quinton, P. Gachet, "Diastol User's Manual- Preliminary Version", IRISA Research Report No. 233, August 1984.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique